Tools, Demos, Posters

Bernard Carré, Houari Sahraoui, Harald Storrle

Proceedings of the joint track "Tools, Demos, and Posters" of ECOOP, ECSA, and ECMFA, 2013

Bernard Carré, Houari Sahroui, Harald Störrle (Eds.)

Tools, Demos, Posters

Proceedings of the Joint Track "Tools, Demos, and Posters" of ECOOP, ECSA, and ECMFA, 2013

Bernard Carré, Houari Sahroui,Harald Störrle (Eds.)

Editors

Harald Störrle
Technical University of Denmark (DTU)
Richard Petersens Plads, 322.024
DK-2800 Kongens Lyngby
hsto@imm.dtu.dk

Bernard Carré
Houari Sahroui

ISBN: 978-87-643-1188-4
Publisher: Technical University of Denmark (DTU)
Printed by DTU Compute
Technical University of Denmark (DTU)
Building 321, DK-2800 Kongens Lyngby
Copenhagen, Denmark
reception@dtu.dk
www.imm.dtu.dk
2013

Preface

We are very pleased to give you the joint tools, demonstrations, and posters of the 2013 European Conferences on Object-Oriented Programming (ECOOP), Software Architecture (ECSA), and Modeling Foundations and Applications (ECMFA), co-located in Montpellier, France.

An amazing array of topics and contributions have been submitted to this track, resulting in 20 tools on demonstration (6 of which can be seen on-line as videos), and 10 posters on display. Contributions have been submitted either straight to this track, from the common Research Project Symposium, or as spin-offs from from full accepted papers, in particular the validated artifacts of ECOOP and the application and tools track of ECMFA. They cover a wide range of topics including integrated development and modeling environments, code analysis and comprehension, domain specific (modeling) languages, refactoring and re-engineering, requirement engineering, software architectures and embedded systems.

We would like to take this opportunity to thank all participants for their contribution to this amazing event.

July 2013

Bernard Carré
Houari Sahroui,
Harald Störrle

Table of Contents

Tool Descriptions

1 Averroes: Whole-Program Analysis without the Whole Program

Karim Ali, Ondrej Lhoták
University of Waterloo, Canada

Averroes is a tool that enables any existing whole-program call graph construction tool (e.g., Soot, Doop, WALA) to generate sound and precise application-only call graphs efficiently without analyzing the library. Averroes generates a placeholder library that over-approximates the possible behavior of the original library, based on the separate compilation assumption (SCA). The placeholder library is constructed quickly and is typically much smaller in size. Typical efficiency improvements of using Averroes are a factor of 4x to 12x in analysis time and 8x to 13x in memory usage. In addition, Averroes makes it easier to handle reflection soundly.

In this demonstration, we would like to show you how to use Averroes to generate the call graphs for some sample programs. We will explain how to configure the various options in Averroes. We will also show how to use the placeholder library to generate the call graph using Soot and Doop.

2 FPLA: A Modeling Framework for Describing Flexible Software Product Line Architecture

Jennifer Pérez Benedí, Jessica Díaz, Juan Garbajosa, Agustin Yagüe
Technical University of Madrid (UPM) - Universidad Politécnica de Madrid
Systems & Software Technology Group (SYST), E.U. Informática, Madrid, Spain

Nowadays, Software Product Line (SPL) engineering has been widely-adopted in software development. SPL features are realized at the architectural level in product-line architecture (PLA) models. This demonstration presents the FPLA Modeling Framework. FPLA offers tool support for (i) modeling PLAs in a graphical way by providing mechanisms to specify the external and internal variability and the different artifacts of the PLA following a view model, (ii) documenting the design decision driving the PLA solution, (iii) tracing PLAs to features allowing change impact analysis, (iv) configuring a specific product architecture, and (v) translating PL specifications into code (automatically) by guarantying the traceability between PLA and code. FPLA has been used for modeling the PLAs from representative software exemplars to industrial projects, such as OPTIMETER, which has been involved in different developments of two ITEA projects (IMPONET and NEMO&CODED) focused on Smart Grids.

3 Characterization of Adaptable Interpreted-DSML

Eric Cariou, Olivier Le Goaer, Franck Barbier, Samson Pierre
Université de Pau

PauWare engine software is a Java API surrounded by an extra tool (the SCXML2PauWare code generator). PauWare engine allows the construction of end-user business applications using UML 2 on one side, Java ME, SE, EE or Android on the other side. PauWare engine enables the visual programming of complex component/service behaviors by specyfying UML 2 State Machine Diagrams. The execution semantics is that of UML 2, which slightly differs from that of Harel's original Statecharts. Developers may either construct behavioral models from UML 2 modeling tools (OMG XMI format), or from the W3C SCXML format, or they may choose to do so from scratch by directly using the PauWare engine API. PauWare engine makes models persistent at runtime and therefore makes them executable by preventing a break or a gap between the models and their incarnation in Java.

4 The GEMOC Initiative: On the Globalization of Modeling Languages

Benoit Combemale, Robert B. France, Jeff Gray, Jean-Marc Jézéquel
University Rennes 1 (France)
Colorado State University (USA)
University of Alabama (USA)

GEMOC is an open initiative that aims to develop breakthrough for software language engineering (SLE) approaches that support global software engineering through the coordinated use of multiple domain-specific languages. GEMOC researchers aim to provide effective SLE solutions to problems associated with the design and implementation of collaborative, interoperable and composable modeling languages.

The GEMOC initiative aims to provide a framework that facilitates collaborative work on the challenges of using of multiple domain-specific languages in software development projects. The framework consists of mechanisms for coordinating the work of members, and for disseminating research results and other related information on GEMOC activities. The framework also provides the required infrastructure for sharing artifacts produced by members, including publications, case studies, and tools.

5 QUIC Graphs: Relational Invariant Generation for Containers

Arlen Cox, Bor-Yuh Evan Chang, Sriram Sankaranarayanan
University of Colorado Boulder, USA

Programs written in modern languages perform intricate manipulations of containers such as arrays, lists, dictionaries, and sets. We present an abstract interpretation-based tool for automatically inferring relations between the sets of values stored in these containers. Relations include inclusion relations over unions and intersections, as well as quantified relationships with scalar variables. We use an abstract domain constructor that builds a container domain out of a Quantified Union-Intersection Constraint (QUIC) graph parameterized by a variety of integer base domains. We demonstrate the application of QUIC graphs on a variety of programs extracted from the Python test suite.

6 GenMyModel : An Online UML Case Tool

Michel Dirix, Alexis Muller, Vincent Aranega
Axellience, Lille, France

Costs and markets lead engineering teams to collaborate from different locations all over the world. Modelling tools are present in development processes to produce complex software and these tools have to be highly collaborative to permit teams to be productive. Axellience tries to resolve issues about distributed collaboration and modelling with GenMyModel.

7 Vasco: A Visual Churn Exploration Tool

Fleur Duseau, Bruno Dufour, Houari Sahraoui
DIRO, Université de Montreal, Canada

Bloat, and particularly object churn, is a common performance problem in modern framework-intensive applications that consists of an excessive use of temporary objects. Temporaries can impose a significant overhead during the execution due to increased initialization costs. Identifying and understanding sources of churn is a difficult and labor-intensive task, despite recent advances in automated analysis techniques. To address this problem, we designed Vasco, a tool that allows users to visually explore how programs use temporaries. Because churn often crosses method and even framework boundaries, Vasco makes it possible to track temporary objects from their creation to their use. Also, since churn often results from multiple methods within a given region collaborating to build complex temporary data structures, Vasco represents program regions explicitly using the Sunburst visual metaphor. Various churn-related metrics can be visualized using this metaphor, thereby allowing users to quickly identify regions that exhibit suspicious behavior.

8 Rational Software Architect Design Manager 4.x

Maged Elaasar
IBM, Canada

Modeling and design management is a key capability of an application lifecycle development of software and systems in an iterative and collaborative way using formalisms that are best suited for these tasks. It also enables management of designs in a shared repository, role based access to designs by stakeholders, parallel development of designs, change and configuration management of designs, linking designs to and previewing of other related lifecycle artifacts (e.g., requirements, tests, change requests), staying informed of design activities by team members, and understanding the impact of changes to/on designs down or up stream. Design management capability has been added to IBMÃs Collaborative Lifecycle Management (CLM) tool offering. The capability is available from a web client, as well as from the Rational Software Architect (RSA DM) modeling tool.

9 BETTY - Behavioural Types for Reliable Large-Scale Software Systems

Simon Gay, Antonio Ravara
Dept. of Computing, University of Glasgow, UK)
CITI and Dept. of Informatics, FCT, Univ Nova de Lisboa)

Modern society is increasingly dependent on large-scale software systems that are distributed, collaborative and communication-centred. Correctness and reliability of such systems depend on compatibility between components and services that are newly developed or may already exist. The consequences of failure are severe, including security breaches and unavailability of essential services. Current software development technology is not well suited to producing these systems, due to the lack of high-level structuring abstractions for complex communication behaviour.

The COST Action Behavioural Types for Reliable Large-Scale Software Systems uses behavioural type theory as the basis for new foundations, programming languages, and software development methods for communication-intensive distributed systems. Behavioural type theory encompasses concepts such as interfaces, communication protocols, contracts, and choreography. As a unifying structural principle it will transform the theory and practice of distributed software development.

10 An MDE Tool-Chain for Pattern-Based S&D Embedded System Engineering

Brahim Hamid, A. Ziani, J. Geisel and C. Jouvray
IRIT (University of Toulouse) and Trialog, France

In our work, we promote a new discipline for system engineering using a pattern as its first class citizen: Pattern-Based System Engineering (PBSE). Therefore, PBSE addresses two kind of processes: the one of pattern development and the one of system development with patterns. To interconnect these two processes we promote a structured model-based repository of patterns and their related models.

This video tutorial3 presents the SEMCO MDE Tool Suite called TERESA and provides guidelines on how to use it to build and to store reusable artefacts (S&D patterns and property models) in the domain of assistance to the pattern-based secure and dependable embedded system engineering. Once the repository is available, it serves an underlying trust engineering process.

11 Assurance Workbench – Business Process based Testing

Deepali Kholkar, Pooja Yelure, Harshit Tiwari
Tata Consultancy Services, India

Design of an effective test suite is the most time consuming task in the software testing process. Business Process models document the operational processes of an enterprise, therefore can be used to derive tests that represent real-life usage scenarios for systems. Our toolset automates generation of end-to-end test cases from Business Process models, using rules annotated on the model to generate relevant test data. Business Rules can be captured separately and are used by the tool to validate the process model for correctness. Test selection using a variety of criteria including structural and weighted coverage helps compact the test suite. Standard coverage techniques such as boundary value analysis and pairwise can be applied to achieve specific coverage.

12 A Comparative Study of Manual and Automated Refactorings

Stas Negara, Nicholas Chen, Mohsen Vakilian, Ralph E. Johnson, Danny Dig
University of Illinois at Urbana-Champaign, USA

Understanding how developers evolve their code is important for researchers, tool builders, and ultimately, developers themselves, who would benefit from improved development practices and tools. Unfortunately, the traditional source of

code evolution data are Version Control System (VCS) snapshots, which are imprecise and incomplete. To overcome these limitations, we developed CodingTracker, a tool that captures fine-grained and precise changes to the code. The captured code changes are so precise that CodingTracker's replayer enables us to reproduce the state of the evolving code at any point in time. Besides code changes, CodingTracker records many other developer actions, for example, invocations of automated refactorings, tests and application runs, interactions with VCS, etc. Our study shows that CodingTracker's data allows researchers to answer questions that could not be answered using VCS data alone. We augmented CodingTracker with an algorithm that infers refactorings from continuous code changes and showed that such inference is precise.

13 Feature-Oriented Programming With Object Algebras

Bruno C.d.S. Oliveira, Tijs van der Storm, William R. Cook, Alex Loh
National University of Singapore
Centrum Wiskunde & Informatica (CWI)
University of Texas, Austin

Object algebras are a new programming technique that enables a simple solution to basic extensibility and modularity issues in programming languages. In this tool demo we will demonstrate the extension of basic object algebras with object algebra combinators to allow feature-oriented programming. The combinators support type safe, extensible, decoration and combination of object algebras. In the demo this is shown using a case study for context-free grammars. Specifically, we will show how this approach enables:

- Combining multiple operations: for instance, parsing and printing of grammars.
- Decoration of operations with aspects like tracing, memoization or fixpoint computation.
- Combining dependent operations: for instance, computing first sets of a grammar requires nullability analysis.

14 A light-weight annotation-based solution to design Domain Specific Graphical Modeling Languages

Francois Pfister, Vincent Chapurlat, Marianne Huchard, Clémentine Nebut
LGI2P, Ecole des Mines d'Alès, Nimes
LIRMM, CNRS, Université Montpellier 2, France

DSML (Domain Specific Modeling Languages) are an alternative to general purpose modeling languages (e.g. UML or SysML) for describing models with concepts and relations specific to a domain. DSML design is often based on Ecore meta-models, which follow the class-relation paradigm and also require defining

a concrete syntax which can be either graphical or textual. In this demo, we focus on graphical concrete syntax, and we introduce an approach and a tool (Diagraph) to assist the design of a graphical DSML. The main points are: non-intrusive annotations of the meta-model to identify nodes, edges, nesting structures and other graphical information; immediate validation of meta-models by immediate generation of an EMF-GMF instance editor supporting multi-diagramming. Diagraph plays the role of an extension to Ecore, and is based on a pattern recognition principle in order to infer most of the concrete syntax.

15 An Eclipse plug-in to link modeling and code proof, AGrUM: ACSL Generator from UML Model

Anthony Fernandes Pires, Thomas Polacsek, Stéphane Duprat
ONERA and Atos Intégration SAS, Toulouse, France

If Model Based Engineering (MBE) supports engineers from the specification to code generation, we want here to bridge the gap between models and static program analysis. Today, static analysis tools prove the source code correctness against a set of properties by using mathematical methods, without executing the program. If these techniques are quite efficient and are used in safety critical contexts (for instance: aeronautic, nuclear, etc.), they are quite difficult to understand and to use. AGrUM is a prototype Eclipse plug-in which purpose is to automatically generate proof properties for C programs from UML State Machine designs to which the programs must conform. These generated properties can then be automatically proved using static analysis tools.

16 Requirements-level migration of legacy systems

Michal Smialek, Wiktor Nowakowski, Norbert Jarzebowski, Kamil Rybinski, Slawomir Blatkiewicz
Warsaw University of Technology, Warsaw, Poland

Novel software development technologies introduce significant changes in system design, delivery and usage patterns. For many legacy applications it means that their further development becomes infeasible due to obsolescence of technologies they use. There thus arises the need for tools supporting automated migration of legacy systems into new paradigms. In this demonstration we will show such a tool suite. Its most important characteristic is that it operates at the level of stories (scenarios) traditionally produced during requirements speciffcation. The tools highly support extracting knowledge about the functioning of legacy applications and storing them in the form of precise requirement-level models. What is more, the tools enable automated transformation of these models into object-oriented code, compliant with new system structure. In the demonstration we present a recovery engine, a requirements-level editor and a transformation engine in the context of a case study example.

17 Verification Condition Generation for Permission Logics with Abstract Predicates and Abstraction Functions

Stefan Heule, Ioannis T. Kassios, Peter Müller, Alexander J. Summers
ETH Zürich, Switzerland

Chalice is a verification tool for concurrent programs that proves that programs are free from data races and deadlocks, do not cause run-time errors, and satisfy assertions provided by the programmer. To verify a program, Chalice generates a verification condition for each method of the program and passes it to an SMT solver. Hence, verification is modular and automatic. Chalice's verification methodology centers around permissions; a memory location may be accessed by a thread only if that thread has permission to do so. We demonstrate Chalice with a focus on its support for data abstraction via abstract predicates and abstraction functions, as explained in our ECOOP paper.

18 Demonstration of a Tool for Consistent Three-way Merging of EMF Models

Felix Schwägerl, Sabrina Uhrig and Bernhard Westfechtel
University of Bayreuth, Germany

Optimistic version control strategies allow for independent modifications of the same software model. As soon as these modifications happen concurrently, three-way merging comes into play. We have developed a formal approach and implemented a tool to meet the specific requirements of merging EMF model versions. Our approach is state-based and advances the state of the art by guaranteeing a consistent merge result. The implementation follows an incremental design: An intermediate structure, the merge model, is alternately modified by the merge tool and by the user until it remains conflict-free. The user interface shows a superimposition of the three versions and provides a wizard for the resolution of context-free and context-sensitive merge conflicts. We demonstrate our tool by means of a UML scenario where two sets concurrent modifications are merged with the help of user interactions.

19 The Requirements Editor RED

Harald Störrle, Maciek Kucharek
Technical University of Denmark (DTU), Denmark

The Requirements Editor (RED) is a tool to support teaching Requirements Engineering graduate courses. The need of tool support is quite obvious, but all the tools on the market covered only a small segment of these techniques, used a

different (and often inconsistent) terminology, and were hard to customize. After several failed attempts to use pre-existing tools, we decided to build our own.

RED is based on Eclipse RCP which makes it relatively stable and comfortable to use. It covers a great variety of RE techniques, from stakeholder analysis and goal modeling via interaction design and classic textual requirements to UML models. Apart from bread-and-butter features like reporting, a help system, and an integrated glossary, RED also provides features specifically designed to improve the learning experience, and research-oriented features not found in conventional tools, such as model fragment folding or inspection support.

20 MetaEdit+: Creating Tool Support for Domain-Specific Modeling Languages

Juha-Pekka Tolvanen
MetaCase, Finland

With MetaEdit+ you can build Domain-Specific Modeling tools and generators - without having to write a single line of code. This demonstration shows how different domain-specific languages (DSLs) can be integrated with a common metamodel and how languages can be created iteratively while auto- matically updating existing models.

21 Really Automatic Scalable Object-Oriented Reengineering

Marco Trudel, Carlo A. Furia, Martin Nordio, Bertrand Meyer
ETH Zurich, Switzerland

AutoOO is a fully automatic migration tool that translates C programs into object-oriented Eiffel projects. The translation applies reengineering heuristics that extract design elements implicit in the C source and render them through Eiffel's object-oriented features. For example, C functions become Eiffel methods and are allocated to classes to achieve proper encapsulation. In this demo, we will demonstrate AutoOO in action on some open-source C programs. The main example will be "xeyes", the widget for the X Windows System showing two googly eyes following the cursor. After translating xeyes from C to object-oriented Eiffel, we will demonstrate how to modify the automatically translated code using Eiffel's standard IDE.

22 A Compositional Paradigm of Automating Refactorings

Mohsen Vakilian, Nicholas Chen, Roshanak Zilouchian Moghaddam, Stas Negara, Ralph E. Johnson

10

University of Illinois at Urbana-Champaign, USA

Making complex code changes is tedious and error-prone. Programming environments provide tool support for various code changes including refactorings. However, studies suggest that programmers greatly underuse automated refactorings and rarely use the ones that automate complex changes. We will demonstrate a refactoring tool that supports a new paradigm of automating refactorings called the compositional paradigm. In this paradigm, the tool designer decomposes a change into smaller ones. The tool automates these smaller changes, and programmers manually compose these changes to perform large changes. In contrast to the the wizard-based paradigm, in the compositional paradigm, programmers perform a complex change in multiple steps rather than a single step. The compositional paradigm provides more control and predictability and reduces the costs of configuring and learning the tool. Our analysis of existing refactoring tool usage data and controlled studies suggest that the compositional paradigm is natural to programmers and outperforms the wizard-based paradigm.

FPLA: A Modeling Framework for Describing Flexible Software Product Line Architecture*

Jennifer Pérez, Jessica Díaz, Juan Garbajosa, Agustín Yagüe

Technical University of Madrid (UPM) - Universidad Politécnica de Madrid
Systems & Software Technology Group (SYST), E.U. Informática, Madrid, Spain
jenifer.perez@eui.upm.es, yesica.diaz@upm.es, jgs@eui.upm.es, agustin.yague@upm.es

1 Introduction

Nowadays, Software Product Line (SPL) engineering [1] has been widely-adopted in software development due to the significant improvements that has provided, such as reducing cost and time-to-market and providing flexibility to respond to planned changes [2]. SPL takes advantage of common features among the products of a family through the systematic reuse of the core-assets and the effective management of variabilities across the products. SPL features are realized at the architectural level in product-line architecture (PLA) models. Therefore, suitable modeling and specification techniques are required to model variability. In fact, architectural variability modeling has become a challenge for SPLE due to the fact that PLA modeling requires not only modeling variability at the level of the external architecture configuration (see [3,4] literature reviews), but also at the level of internal specification of components [5]. In addition, PLA modeling requires preserving the traceability between features and PLAs. Finally, it is important to take into account that PLA modeling should guide architects in modeling the PLA core assets and variability, and in deriving the customized products. To deal with these needs, we present in this demonstration the **FPLA Modeling Framework.**

FPLA Modeling Framework offers tool support for (i) modeling PLAs in a graphical way by providing mechanisms to specify the external and internal variability and the different artifacts of the PLA following a view model, (ii) documenting the design decision driving the PLA solution, (iii) tracing PLAs to features allowing change impact analysis, (iv) configuring a specific product architecture (PA), and (v) translating PL specifications into code (automatically) by guarantying the traceability between PLA and code. FPLA has been used for modeling the PLAs from representative software exemplars [6] to industrial projects, such as OPTIMETER [7] which has been involved in different developments of two ITEA projects (IMPONET and NEMO&CODED) focused on Smart Grids. Specifically, to illustrate the use of FPLA, this demonstration shows several snapshots of one of the developments of OPTIMETER (see Fig. 1). OPTIMETER consisted of the development of a SPL for metering management systems in electric power networks [7]. Metering management

* INNOSEP (TIN200913849), IMPONET (ITEA 2 09030, TSI-02400-2010-103), iSSF (IPT-430000-2010-038), NEMO&CODED (ITEA2 08022, IDI-20110864) and ENERGOS (CEN-20091048).

systems capture and manage meter data from a large number of distributed energy resources, load these data in a database, support data querying and processing, and provide these data to other systems for billing, forecasting or purchasing.

2 FPLA Modeling Framework

FPLA supports Model-Driven Development (MDD) of SPLs by modeling feature models (adopting feature-oriented analysis), PLAs, traceability from features to PLA through architectural design decisions (ADD), and PA configurations; and also by generating AspectJ code from models. FPLA modeling is supported by a 4+1 view model that consists of the Core, Variability, Derivation, Product, and PLAK (Product-Line Architectural Knowledge) views. Fig. 1 shows the models and views of OPTIMETER. The feature model describes the features of OPTIMETER (see A). The core view (see B) allows configuring the common PLA for all the products of the SPL. This view is supported by components, such as the *Database Manager*, and **Plastic Partial Components** (PPCs) [8], such as the *Data Querying*. PPCs allows specifying the internal variation of components, in such a way that part of its behavior corresponds to the core functionality of a SPL and part of its behavior is specific of a product or set of products from that SPL. The variability view (see C) supports modeling: the **external variability** by adding and removing attachments among components, and the **internal variability** of PPCs. The PPCs *DataQuerying* and *Data Loader* implement the variability for the different data storing technologies. The variability of a PPC is specified using *variability points* (see the AVP *clustering)* which hook fragments of code to the PPC known as *variants* (see the variants *HadoopMAP/REDUCE* and *RAC)*. Finally, *weavings* specify where and when to extend PPCs through the use of variants (see C). The PLAK view (see D) allows one to trace features to PLA through design decisions and to analyze the impact of changes. The derivation and product views (see E) allow architects to select the variants of a specific product of the SPL and to generate its PA. In this case, two specific products were configured and their code skeletons were automatically generated by automatically composing the required source-code from external sources. FPLA is a plug-in of Eclipse that has been developed by using the infrastructure provided by the Eclipse Modeling Framework for the modeling support, and the Epsilon Generation Language of the Epsilon Generative Modeling Technologies research project for the model-to-code transformation. A complete video demonstration is available at https://syst.eui.upm.es/FPLA/examples.

REFERENCES

1. Pohl K., Böckle G., van der Linder F.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag (2005)
2. Schmid, K., Verlage, M.: The economic impact of product line adoption and evolution. IEEE Softw. 19(4) pp. 50–57 (2002)
3. Schaefer I., et al.: Software diversity: state of the art and perspectives. International Journal on Software Tools for Technology Transfer 14(5), Springer-Verlag pp. 477-495 (2012)

4. Chen, L., Ali Babar, M., Ali, N.: Variability management in software product lines: a systematic review. In Proc. of the Int. Software Product Line Conference, pp. 81-90 (2009)
5. Bachmann F., Bass L.: Managing variability in software architectures. In Proc. of the symposium on Software reusability. New York, NY, USA: ACM, pp. 126–132. (2001)
6. Díaz, J., Pérez, J., Garbajosa, J., & Wolf, A: Change impact analysis in product-line architectures. In Software Architecture, volume 6903 of LNCS, Springer pp. 114-129 (2011).
7. Díaz, J, Pérez J, Garbajosa J, Yagüe A.: Change-Impact driven Agile Architecting. In Proc. of the 46th Hawaii Int. Conference on System Sciences (HICSS '13), IEEE (2013)
8. Pérez, J., Díaz, J., Soria, C. C., Garbajosa, J.: Plastic partial components: A solution to support variability in architectural components. In Proc. WICSA/ECSA 2009, IEEE (2009)

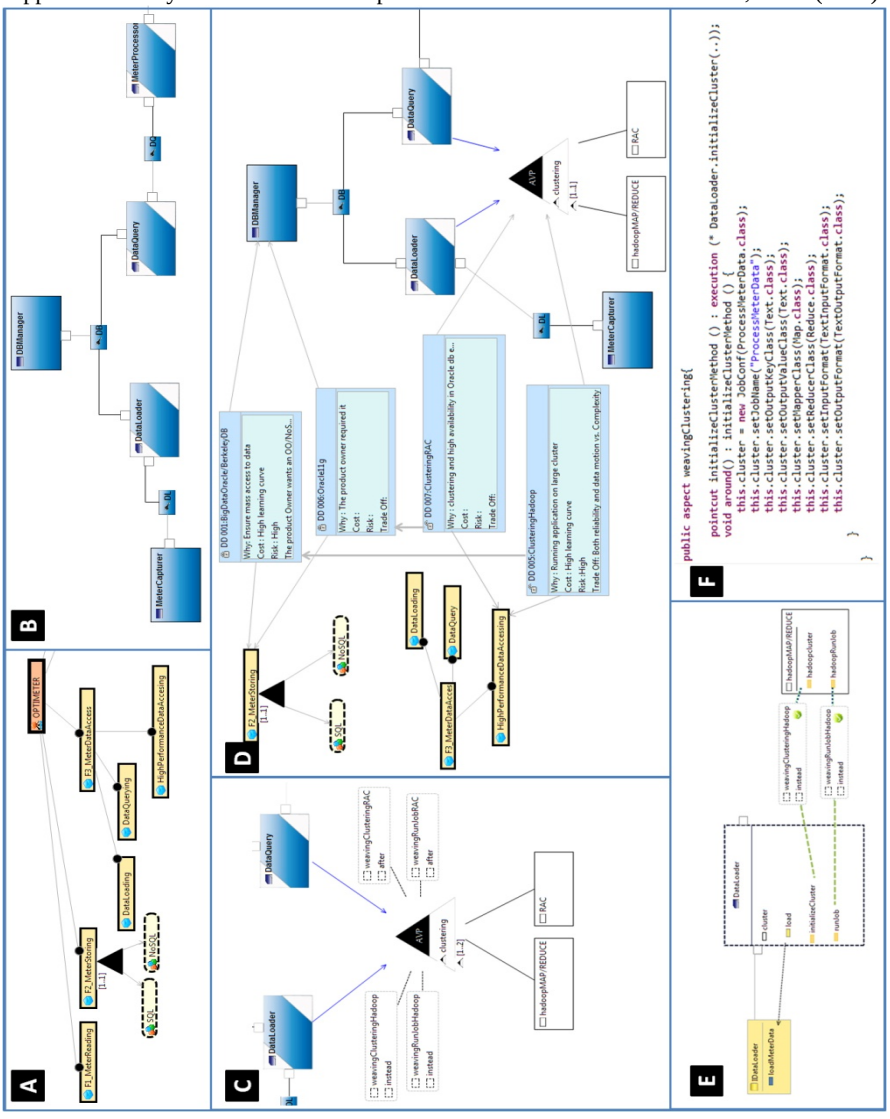

Fig. 1. FPLA Modeling Framework – OPTIMETER project

GenMyModel : An Online UML Case Tool

Michel Dirix, Alexis Muller, and Vincent Aranega

Axellience 59000 Lille - France
{michel.dirix,alexis.muller,vincent.aranega}@axellience.com

Abstract. Costs and markets lead engineering teams to collaborate from different locations all over the world. Modelling tools are present in development processes to produce complex software and these tools have to be highly collaborative to permit teams to be productive. Axellience tries to resolve issues about distributed collaboration and modelling with GenMyModel.

Keywords: UML, Collaboration

1 Introduction

Building complex software is a collaborative activity where modelling holds an important place. Since a decade, outsourcing and offshoring projects or part of them have become regular practices in software industry. These ones can have different reasons like the need to cut down costs or to explore new markets. Collaboration between project team members suffers from resulting geographical distribution.

Software practitioners usually draw diagrams when they design new applications, maintain existing ones or discuss with their clients using modelling tools [1]. Besides, models are important to manage knowledge and provide an efficient support to coordinate activities when the projects aim at producing complex software. It is important that modelling tools intent to support collaboration in the context of distributed teams, that is to say, to provide similar working conditions as co-located settings. Whitehead shows that the trend concerning collaboration support is the arrival of web-based tools in every phase of software development [2]. Their main advantage is that they do not require any installation or configuration: teams are quickly ready to work.

Axellience was launched from this observation and had the support of INRIA in April 2012. Now, Axellience offers GenMyModel which is an online UML and generator tool. Axellience has began to communicate about a beta-version of GenMyModel since January 2013 and there are already nearly a thousand of users spread over in more than 70 countries. Some models already reach more than 50 classes elements and includes several hundreds of model elements.

2 GenMyModel

The promises of SaaS (Software as a Service) is to allow users to access to services without installation or configuration. Users find the same environment

no matter where they connect from. In addition, SaaS applications can be used by users from anywhere, anytime, from any device and operating system, and users do not have to update their application. So, GenMyModel is online, in SaaS mode and its Graphical User Interface is shown in the figure 1. The intention of GenMyModel is to accelerate the modelling phase. Users are ready to work quickly and they access GenMyModel and the models from anywhere at anytime. For example, an architect can begin to model at his/her office, continue at home and present the result to the client the day after.

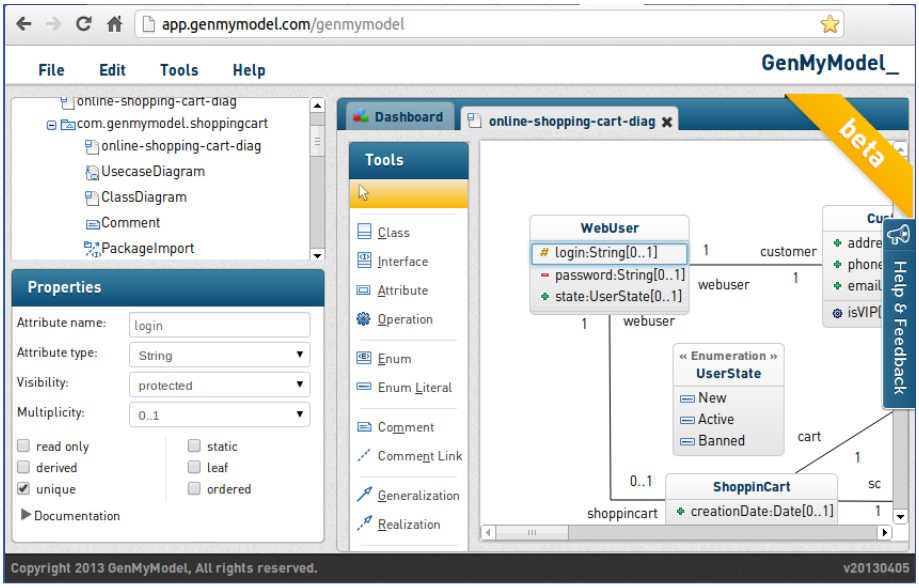

Fig. 1. GenMyModel Graphical User Interface

Today, GenMyModel supports Class Diagrams and Use Case. Others are in development. From Class Diagrams, GenMyModel offers its users the possibility to generate the code to multiple targets like Java, Java-JPA, SQL. Others will be added. In addition, the code is preserved throughout different generations. So, users can edit their code, model again, re-generate without code loss. Nevertheless, in most cases, a generator is not completely adapted to users needs. GenMyModel will add a generator upload service where users would add their own generators.

Another benefit of the SaaS applications is the simplification of real-time collaboration [3]. Real-time collaboration is an ongoing work into GenMyModel and a first version will be available soon. We are working on the model synchronization between users and the rights management. In the longer term, we have to study what kind of information is necessary to display to users in order to facilitate collaboration.

Future work will be on model repositories too. We plan to propose model repositories with versioning facilities. Thereby, users will can return to next version of their model.

One of the aims of GenMyModel is to offer a platform for specific needs or experimentations. For that, usage of GenMyModel will be free for public projects, allowing users to easily experiment and share their work. Additionally, the access to the model repositories will be granted to other tools though an API. This API will contain the generator upload service. By opening this API, Axellience wants to continue the collaboration with research laboratories where Axellience is already engaged.

3 Demonstration content

1. How to access to GenMyModel;
   We will create a GenMyModel user in less than 30 seconds and will access to GenMyModel
2. Simple model creation;
   We will present the GenMyModel Graphical User Interface and will create a class diagram.
3. Code generation;
   From a simple model, we will generate associated code using the Java-JPA generator. This generation will be uploaded into a Github repository and we will show the generated code and annotations.
4. Model modification and code regeneration;
   We will edit the generated code from a code editor tool in the Cloud. Then, we will edit our model to add new properties and we will generate the code again to show that new generations do not modify the code edited before.
5. Current state of collaboration.
   To conclude, we will show our ongoing work on collaboration in editing the same model with two users.

References

1. Grossman, M., Aronson, J.E., McCarthy, R.V.: Does uml make the grade? insights from the software development community. Inf. Softw. Technol. **47**(6) (April 2005) 383–397
2. Whitehead, J.: Collaboration in software engineering: A roadmap. In: Future of Software Engineering, 2007. FOSE'07, IEEE (2007) 214–225
3. Brunelière, H., Cabot, J., Jouault, F.: Combining Model-Driven Engineering and Cloud Computing. In: Modeling, Design, and Analysis for the Service Cloud - MDA4ServiceCloud'10: Workshop's 4th edition (co-located with the 6th European Conference on Modelling Foundations and Applications - ECMFA 2010), Paris, France (June 2010)

Vasco: A Visual Churn Exploration Tool

Fleur Duseau, Bruno Dufour, and Houari Sahraoui

DIRO, Université de Montréal, Canada
{duseaufl,dufour,sahraouh}@iro.umontreal.ca

Abstract. We describe a tool to visualize the usage of temporary objects during the execution of a program. This tool is designed to help programmers locate and remove sources of *object churn* in their application, a common performance problem due to the excessive use of temporaries.

Keywords: visualisation, framework-intensive applications, object churn, dynamic analysis, execution traces

1 Introduction

Bloat, and particularly *object churn*, is a common performance problem in modern framework-intensive applications. Object churn consists of an excessive use of temporary objects. Temporaries can impose a significant overhead during the execution due to increased initialization costs. Identifying and understanding sources of churn is a difficult and labor-intensive task, despite recent advances in automated analysis techniques.

To address this problem, we designed Vasco [4], a tool that allows users to visually explore how programs use temporaries. Because churn often crosses method and even framework boundaries, Vasco makes it possible to track temporary objects from their creation to their use. Also, since churn often results from multiple methods within a given region collaborating to build complex temporary data structures, Vasco represents program regions explicitly using the Sunburst visual metaphor [5]. We then present various churn-related metrics using this metaphor. This allows users to quickly identify regions that exhibit suspicious behavior.

2 Design Principles

The design of Vasco was guided by three main principles:

- *scalability*: the large amount of data to visualize requires a judicious use of the screen real estate as well as abstractions.
- *low cognitive effort*: intuitive metaphors help to reduce the cognitive effort required to perform a task. We use natural abstractions that are easily perceived (e.g., the size and color of a visual entity) to draw the attention of the user to important parts of the system under consideration.
- *smooth transitions and interactions*: leaving visual clues when performing transitions and proper placement of the required information make the tool more intuitive and efficient.

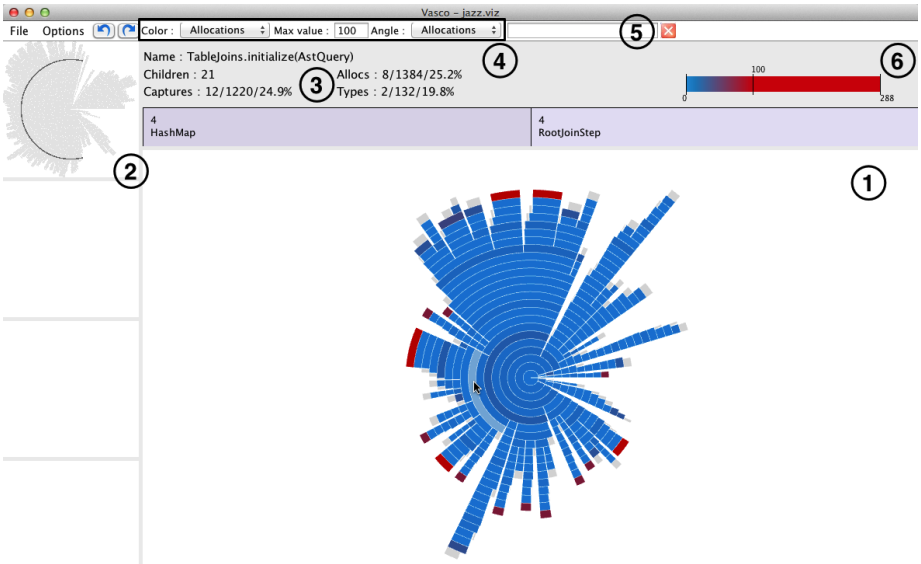

Fig. 1. Areas of the visualization tool

3 Input Data

Vasco visualizes execution data extracted from dynamic traces collected using the Jinsight tool [2]. For scalability, we use Calling Context Trees (CCTs) [1] to represent calls. CCTs represent invocations as a tree where method invocations resulting from the same sequence of calls (i.e., the same call stack at the time of invocation) are merged into a single node. Temporary objects are identified using a blended escape analysis [3]. This analysis identifies objects are *captured* or *escaping* from a subtree of the CCT. Captured objects and data structures correspond to likely temporaries.

The color and the angle spanned by a node in the Sunburst metaphor can be used to display information. Vasco supports four churn-related metrics that can be mapped to each attribute: number of allocated types, number of allocated objects, number of captured objects, number of allocated objects that are likely temporaries.

4 Graphical Interface

The typical usage scenario in our approach is an iterative process in which the user first attempts to locate the most significant source of churn, removes it from the view and repeats the process until no sources of churn remain.

Figure 1 shows a screenshot the tool. The main view (area ①) displays the CCT as a Sunburst, with the root in the center. The use can navigate an filter the main view by selecting a subtree to display. A desaturated representation of

the view prior to the selection is then added to the context sidebar (area ②). The information panel (area ③) displays the most useful data about the entity currently under the mouse pointer. Vasco computes several metrics that provide different views of the data. The mapping of the metrics can be changed directly from the menu bar (area ④). Metrics used for the color attribute and the arc size can be selected independently. For the color mapping, the value corresponding to the maximal color can also be selected by the user by interacting with the color gradient directly (area ⑥). Users can enter a search string that will cause all matching nodes to be selected (Figure 1 area ⑤). Nodes are matched based on class and method names.

5 Conclusions and Future Work

Vasco is an interactive visualization approach that helps developers to locate and understand significant sources of churn in their applications. It leverages dynamic data generated by an existing blended escape analysis tool. Vasco relies on known preattentive perception principles in order to reduce the cognitive effort required to perform the visual analysis task.

As future work, we want to support a richer set of interactions, such as the ability to search for invocations allocating or capturing the same types of instances. We would also like to integrate Vasco in Eclipse. We also plan to extend our approach to other key components of bloat and, more generally, other problems and program understanding tasks that require a precise exploration of the behavior. For example, Vasco could be used to track the flow of data within a complex application.

References

1. Ammons, G., Ball, T., Larus, J.R.: Exploiting hardware performance counters with flow and context sensitive profiling. In: Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI). pp. 85–96 (1997)
2. DePauw, W., Jensen, E., Mitchell, N., Sevitsky, G., Vlissides, J., Yang, J.: Visualizing the execution of Java programs. In: Software Visualization: State of the Art Survey, LNCS 2269 (2002)
3. Dufour, B., Ryder, B.G., Sevitsky, G.: A scalable technique for characterizing the usage of temporaries in framework-intensive Java applications. In: Proceedings of the International Symposium on the Foundations of Software Engineering (FSE) (2008)
4. Duseau, F., Dufour, B., Sahraoui, H.: Vasco: A visual approach to explore object churn in framework-intensive applications. In: Proceedings of the International Conference on Software Maintenance (ICSM). pp. 15–24 (2012)
5. Stasko, J., Zhang, E.: Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In: Proceedings of the IEEE Symposium on Information Visualization (InfoVis). pp. 57–65 (2000)

An open and distributed framework for designing Domain Specific Graphical Modeling Languages

[1]François Pfister, [1]Vincent Chapurlat, [2]Marianne Huchard, [2]Clémentine Nebut

[1]LGI2P, Ecole des Mines d'Alès, site de Nîmes, Parc Scientifique G. Besse, 30000 Nîmes, France {forename.lastname}@mines-ales.fr
[2]LIRMM, CNRS – Université Montpellier 2, 161 rue Ada, 34095 Montpellier Cedex 5, France {lastname}@lirmm.fr

Abstract. DSML (Domain Specific Modeling Languages) are an alternative to general purpose modeling languages (e.g. UML or SysML) for describing models with concepts and relations specific to a domain. The design of DSML requires defining a graphical notation over an abstract meta-model. In this demo, we introduce an approach and a tool (Diagraph) to assist the design of a graphical DSML. The main points are: non-intrusive annotations of the meta-model to identify nodes, edges, nesting structures and other graphical information; immediate validation of meta-models by immediate generation of an EMF-GMF instance editor supporting multi-diagramming. Diagraph is based on a pattern recognition principle in order to infer most of the concrete syntax.

Keywords: dsml, mde, meta-model, language workbench, graphical concrete syntax

1 Introduction

Practitioners who want to model technical or socio-technical systems start with a class-relation formalism, either by extending UML, or using MOF, an initial class formalism, to define the concepts which were previously unavailable, so as to obtain a new language tailored to their field. Such a work has two major phases: first, defining the abstract syntax with the use of a class diagram, and second, defining the concrete syntax, which specifies the form of (textual or graphical) statements that conform to the abstract syntax.

In this demo we are interested in graphical concrete syntaxes. There is no consensus (as this exists with MOF for abstract syntaxes or EBNF for textual concrete syntaxes) about a description language for graphical concrete syntaxes. Indeed, designing and implementing a graphical notation is a strenuous activity requiring significant expertise, both in its semiotic and cognitive concerns, and in its technical and operational aspects. We demonstrate a process and a tool for agile development of graphical modeling languages on the top of Ecore.

So to be able, in an agile way, to design domain specific tailored languages, our framework, named Diagraph, allows to design simultaneously their abstract syntax

(meta-model) and their concrete syntax (notation and diagramming). Numerous solutions exist, but none of them satisfies the needs we will detail in the demo.

We aim to integrate Diagraph into the Eclipse ecosystem, and to comply with the current standards of Model Based Engineering (MBE). Our intention is to reuse already available components. Thus, we designed Diagraph as a technical overlay over GMF [2], which is powerful but overly complex for the end user. In addition, we propose a methodology, which lacks in most of the existing propositions. In our approach, defining a new graphical modeling language is made by annotating the concrete syntax on the classes that make up the abstract syntax, by the mean of our Diagraph description language. These annotations, that have been automatically generated by an integrated wizard, and amended by the human expert, associate the concrete syntax to the abstract constructs. The resulting target modeling language is defined in one sole artifact, by this principle, which is that of a grammar.

2 Related work

Many frameworks are able to generate graphic editors from which we can create models, instances of a given meta-model. The generation of these editors takes at the input the given meta-model on one hand, and manual parameters given by the modeling expert on the other hand. The degree of automation of the generation process remains a challenge.

GMF [2] is a framework based on a mapping between MOF and a graph drawing engine, as a part of Eclipse Emf-Ecore stack [1]. This framework is powerful, but poorly documented, and therefore requires a huge technical expertise, this results in a steep learning curve. MetaEdit+ [7] is not based on the Emf-Ecore stack, but on a specific meta-meta-model named GOPRR (Graph, Object, Property, Role and Relationship). GME (Vanderbilt University) is based on MS Component Object Model technology. Microsoft Dsl Tools has a proprietary meta-meta-model, while XMF Mosaic's is based on an infrastructure named XCore. Obeo Designer [3] is a modeling environment based on the notion of points of view. It is a component of the Eclipse platform, and is based on EMF and GMF-Runtime. Obeo Designer and MetaEdit+ are commercial tools that are split in two different parts: a workbench, a tool for designing modeling languages and a modeler, a tool for using modeling languages.

Eugenia[4], a free and open tool based on Emf, proposes to annotate the meta-model with concrete syntax statements, to generate the GMF artifacts, avoiding the user to deal with the poorly designed GMF workflow. This latter tool lacks of an integrated support of multiple points of view, required for large meta-models in the true life.

3 The Diagraph proposal

As it results from the above survey, several limitations of the existing offer lead us to design Diagraph, a new language and framework. As Eugenia[4], we adopt the

principle of annotations, but in an improved way, enhanced by a semi-automated mechanism that will infer a huge part of the graphical notation. Furthermore, the tool is a part of an integrated framework which acts as core engine within an open and world-wide distributed environment [5, 6] dedicated to the design of visual modeling languages. Diagraph offers at the same time:

- An easy to use solution
- A graphical notation inference mechanism, based on pattern recognition,
- A native support of the multi-view paradigm,
- A native and easy support of nested and affixed nodes,
- Integration in the Eclipse-OSGI ecosystem, which is a de facto standard in the Model Based Engineering field,
- An open technology [5], with a published meta-model (MOF compliant), that defines a pivot concept of diagramming, independent of any platform, in one hand, and targets several platforms on the other hand (GMF runtime and Graphiz Dot at the moment),
- A really usable and regularly updated tool.
- A shared and collaborative repository [6] of visual modeling languages, coming as use cases, bundled with their graphical editors, and several examples for each language.

4 References

1. Budinsky, F. et al.: Eclipse Modeling Framework. Pearson Education (2003).
2. Gronback, R.: Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Addison-Wesley Professional (2009).
3. Juliot, E., Benois, J.: Viewpoints creation using Obeo Designer or how to build Eclipse DSM without being an expert developer?, http://www.obeo.fr/resources/WhitePaper_ObeoDesigner.pdf.
4. Kolovos, D. et al.: Taming EMF and GMF Using Model Transformation. In: Petriu, D. et al. (eds.) Model Driven Engineering Languages and Systems. p. 211--225 Springer, Berlin / Heidelberg (2010).
5. Pfister, F.: Diagraph, a Framework over EMF and GMF to automate the design of graphical Domain Specific Modeling Languages, http://code.google.com/p/diagraph/.
6. Pfister, F.: OpenDSML: An Open Framework for Domain Specific Graphical Modeling Languages, http://www.opendsml.org.
7. Tolvanen, J.-P., Kelly, S.: Integrating Models with Domain-Specific Modeling Languages. Systems Programming Languages and Applications: Software for Humanity (formerly known as: OOPSLA). (2010).

An Eclipse plug-in to link modelling and code proof

AGrUM: ACSL Generator from UML Model

Anthony Fernandes Pires[1,2], Thomas Polacsek[1], and Stéphane Duprat[2]

[1] ONERA, 2 avenue Edouard Belin,
F31055 Toulouse, France
[2] Atos Intégration SAS, 6 impasse Alice Guy, B.P. 43045,
31024 Toulouse cedex 03, France

Abstract. If Model Based Engineering (MBE) supports engineers from the specification to code generation, we want here to bridge the gap between models and static program analysis. Today, static analysis tools prove the source code correctness against a set of properties by using mathematical methods, without executing the program. If these techniques are quite efficient and are used in safety critical contexts (for instance: aeronautic, nuclear, etc.), they are quite difficult to understand and to use. AGrUM is a prototype Eclipse plug-in which purpose is to automatically generate proof properties for C programs from UML State Machine designs to which the programs must conform. These generated properties can then be automatically proved using static analysis tools.

Keywords: Model Based Engineering, formal methods, code verification, UML, State Machine, Eclipse, plug-in

1 Introduction

In software development, verification activities have a significant cost. In the sixties, Hoare [2] was already reporting that over half of software development time was dedicated to program testing. Today, in embedded software development, we notice at Atos that the cost of verification activities can sometimes reach 60% of the project workload.

Formal methods are mathematically-based techniques, for instance, formal logic, model checking or discrete mathematics. They allow performing rigorous verification tasks during software development, to enable a more effective identification of software defects and to reduce verification costs. They are already applied in industrial context [4] [3]. But one of their major inconvenient is that they are quite difficult to understand for non experts. These methods have mathematical background which can represent a significant learning cost for beginners or just repulse them. So, how to easily bring the power of formal methods to people who do not understand them?

Model Based Engineering (MBE) allows users to model software in an easy way dealing with systems complexity and the extended enterprise approach of

nowadays projects. It also allows taking advantage of the modelling to generate code, documentation or tests and so to support developers all along the development process. Following this spirit, we propose to provide a way to use the design of software to verify its implementation concealing the use of formal methods from users.

2 Motivations

The Unified Modeling Language (UML) is widespread and it is currently used in industry development teams. We propose to use a particular subset of UML based on the subset defined in [1] for the specification of embedded software. The subset we are interested in deals with the behavioural representation of software, realized through UML state machine diagram. We define a first subset to conduct our work, limited to very simple concepts (for instance, no state hierarchy, no effects on transition, etc). In addition, we consider these state machines are meant to be driven by a clock and to do a certain number of actions at each clock tick. In this way, we limited the possible events handled by the state machine to the completion event (no event) and to the clock tick.

Our goal is to verify source code behaviour according to its UML state machine design. For the verification task, we are interested in static analysis. It allows the detection of bugs and the verification of properties on a program without executing it. We propose to automatically generate annotations from the model into the code. These annotations represent the behavioural properties of the model. They will be automatically verified by a static analysis tool and so allows users to automatically verify their implementation behaviour.

We can compare our method with a code generation method. At a technical level, automation of properties generation for verification purpose is similar to automation of code generation. But, placed in a certification context like DO-178C for the aeronautical domain, the qualification constraints of a verification tool are much lighter than those of a code generator tool. If the verification tool fails, it does not introduce errors in the target software while a code generator might. A code generator must be qualified at least at the same level of criticality than the target software; it is not the case for a verification tool.

3 The tool

AGrUM [3] is an Eclipse plug-in prototype to automatically generate proof properties from an UML State Machine-based design to C code. Our plug-in takes advantage of Eclipse-based tool as Papyrus MDT [4] for the design modelling of the software.

As a first prototype, the UML state machine is implemented following our own created pattern. The proof properties are generated as annotations in ANSI/ISO C Specification Language (ACSL). ACSL is a specification language to

[3] AGrUM for ACSL Generator from UML Model

[4] `www.eclipse.org/papyrus/`

express behavioural properties on C code. It is based on first order logic and allows specifying function contracts, invariants, variants, etc. They are represented as comments in C code, using specific tags and they are without side effects on the program. These annotations can then be proved by the Frama-C [5] framework, which is an open-source tool for the static analysis of C program.

The use of the plug-in is very simple. The user designs its software behaviour as state machines using the Papyrus model editor. He just has to select its state machine in the model explorer, right click on it and select the Generate command. Then, he chooses the C file to annotate and if its code pattern conforms to the state machine and the model respects the subset, the plug-in automatically annotates the C file.

The AGrUM project is store in Eclipse Labs and it is available at:
`http://code.google.com/a/eclipselabs.org/p/agrum/`

Users can find the source code, an update site to automatically install the plug-in in an Eclipse SDK 3.7, information, example and videos on its use. Note that this plug-in is fully compatible with TOPCASED [6] 5.2 or later.

References

1. Fernandes Pires, A., Duprat, S., Faure, T., Besseyre, C., Beringuier, J., Rolland, J.F.: Use of modelling methods and tools in an industrial embedded system project : works and feedback. In: ERTS. France (2012)
2. Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM 12(10), 576–580 (1969)
3. Pariente, D., Ledinot, E.: Formal verification of industrial c code using frama-c: a case study. Formal Verification of Object-Oriented Software (2010)
4. Souyris, J., Wiels, V., Delmas, D., Delseny, H.: Formal verification of avionics software products. In: Cavalcanti, A., Dams, D. (eds.) FM 2009: Formal Methods, Lecture Notes in Computer Science, vol. 5850, pp. 532–546. Springer Berlin Heidelberg (2009), `http://dx.doi.org/10.1007/978-3-642-05089-3_34`

[5] `frama-c.com`
[6] Toolkit in OPen-source for Critical Application and SystEms Development,`www.topcased.org`

Demonstration of a Tool for Consistent Three-way Merging of EMF Models

Felix Schwägerl, Sabrina Uhrig and Bernhard Westfechtel

University of Bayreuth, Universitätsstr. 30, 95440 Bayreuth, Germany
{felix.schwaegerl,sabrina.uhrig,bernhard.westfechtel}@uni-bayreuth.de

Abstract. Version control systems have become indispensable in contemporary software development. Optimistic strategies allow for independent modifications of the same software artifact, e.g. model. As soon as these modifications happen concurrently, three-way merging comes into play. Merging tools specific to the requirements of model-driven development are urgently needed. We demonstrate a three-way merge tool for EMF models which supports the user committing his decisions in order to resolve merge conflicts. The user interface consists of an editor that shows the superimposition of the three input versions and a wizard that guides the user through the resolution of pending merge conflicts. In our example scenario we merge concurrent modifications on a medium-sized UML model.

1 Background

Inadequate version control has been identified as a major obstacle to the application of model-driven software engineering. In particular, sophisticated support for *merging model versions* is urgently needed. Line-oriented or structure-oriented (e.g. XML-based) merging tools do not address crucial requirements concerning the consistency of merge results and the detection and resolution of merge conflicts to a sufficient extent.

In [4], we have developed a formal approach to state-based three-way merging of model versions in the *Eclipse Modeling Framework* (EMF [3]). It allows to detect and resolve *context-free conflicts* occurring on the same structural feature of some EMF object, as well as *context-sensitive conflicts* considering changes of different structural features of potentially different objects. Our approach advances the state of the art by guaranteeing a merge result that is consistent with the structural constraints imposed by EMF and by allowing for merging objects from different classes. We only require that the three models are instances of the same Ecore model. The implementation follows an incremental design allowing to pause or revert merge decisions at any time.

After a brief tool overview in Section 2, we present our demonstration plan in Section 3. We conclude with installation instructions and a web link to the screencasts.

2 Tool Overview

Our merge tool itself is based on EMF; the *BTMerge* metamodel defines the structure of *merge models* which represent the superimposition of the three EMF model versions involved in a merge. As shown in Figure 1, merging with our tool is a three-phase process:

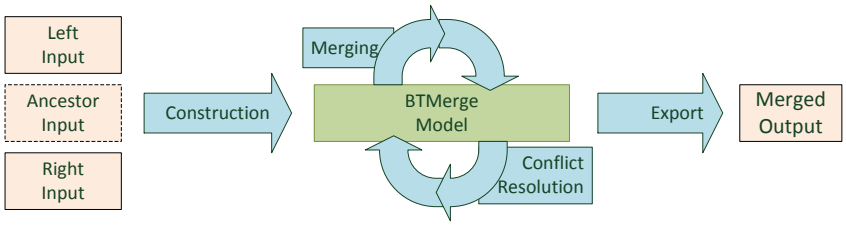

Fig. 1. Conceptual overview on our merge tool.

First, the merge model is created (*construction*). Corresponding EMF objects are identified either by unique identifiers (*UUID*s) or by means of a matching algorithm, e.g. the *EMF Compare match engine* [1]. The second phase, *merging*, follows an incremental design: The preliminary merge model is modified alternately by the merge algorithm and the user; the merge algorithm applies *merge rules* which can either be applied automatically or require user interaction in case of *conflicts*. Only after all conflicts are resolved, the merge model is ready to be *exported* back into an EMF instance.

The user interface, the *resolution tool* (cf. screenshot Fig. 2), is based on a generated EMF tree editor and allows the user to communicate resulution decisions for specific conflicts during the second phase. The *main* editing view (right top) shows the containment tree of the model versions in multiple columns. Below, a modified *properties* view reflects values of structural features of the superimposed object selected in the main view. Conflicting objects and values are marked by icon overlays. The *conflicts* view located at the left bottom outlines pending merge decisions. The user can resolve a conflict by double-clicking on it, which opens a wizard that will describe the conflict and propose several resolution methods. After resolution by the user, the next merge increment is performed automatically until the *merging* phase terminates.

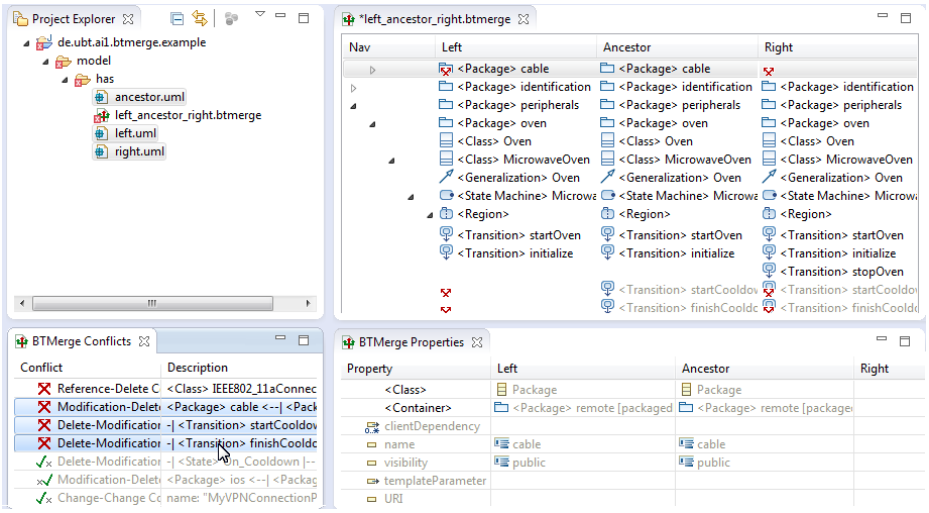

Fig. 2. A screenshot of the resolution tool with the “Home Automation System” example.

3 Demonstration Plan

We apply our merge tool to a medium-sized scenario ("Home Automation System") where three revisions of the same UML [2] model, a common *ancestor* and two alternative versions (*left* and *right*), which result from concurrent modifications, are merged. The plan is accompanied by our screencasts (web link, see Section 4).

1. **Applying Concurrent Modifications**. By means of the UML tree editor, the following conflicting modifications are applied to two copies of the same model:
   (a) Renaming of a class `VendorVPNConnectionProvider` to `MyVPNConnectionProvider` (left) vs. `YourVPNConnectionProvider` (right).
   (b) Insertion of a generalization to class `IEEE802_11aConnector` (left) vs. deletion of the corresponding class (right).
   (c) Visibility change of package `ios` (left) vs. deletion of that package (right).
2. **Merging**. An initial merge model is created from the three versions using the provided context menu entry. In subsequent increments, the following conflicts are reported to the user and resolved by means of the resolution wizard:
   (a) *Change-Change Conflict* on the feature `name` of `VendorVPNConnectionProvider`. The user selects value `MyVPNConnectionProvider` (left).
   (c) *Reference-Delete Conflict* on class `IEEE802_11aConnector`. The user selects "reference" (left) and discards the deletion (right).
   (b) *Modification-Delete Conflict* on package `ios`. The user applies the deletion (right), discarding the modification (visibility change, left). The resolution of this conflict leads to the automatic resolution of related conflicts caused by it.
3. **Result**. The merge output is a valid EMF instance and contains all modifications that have been selected by the user. Furthermore, non-conflicting changes have been applied (e.g. inside the `MicrowaveOvenControl` state chart).

4 Installation Instructions and Screencasts

Our software is available on an update site[1] and can be installed as plug-ins into a clean *Eclipse Modeling Tools* distribution (Juno or higher). Screencasts demonstrating both the installation and the usage of our tool are provided on our web pages[2].

References

1. Brun, C., Pierantonio, A.: Model differences in the Eclipse Modelling Framework. UPGRADE IX(2), 29–34 (Apr 2008)
2. OMG: OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.3. OMG, Needham, MA, formal/2010-05-05 edn. (May 2010)
3. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF Eclipse Modeling Framework. The Eclipse Series, Addison-Wesley, Upper Saddle River, NJ, 2nd edn. (2009)
4. Westfechtel, B.: Merging of EMF models: Formal foundations. Software and Systems Modeling p. 32 p. (Oct 2012), http://dx.doi.org/10.1007/s10270-012-0279-3, Online First

[1] http://btn1x4.inf.uni-bayreuth.de/btmerge/update/
[2] http://btn1x4.inf.uni-bayreuth.de/btmerge/screencasts/

Requirements-level migration of legacy systems

Michał Śmiałek, Wiktor Nowakowski, Norbert Jarzębowski, Kamil Rybiński, Sławomir Blatkiewicz

Warsaw University of Technology,
Warsaw, Poland
smialek@iem.pw.edu.pl

Abstract. Novel software development technologies introduce significant changes in system design, delivery and usage patterns. For many legacy applications it means that their further development becomes infeasible due to obsolescence of technologies they use. There thus arises the need for tools supporting automated migration of legacy systems into new paradigms. In this demonstration we will show such a tool suite. Its most important characteristic is that it operates at the level of stories (scenarios) traditionally produced during requirements specification. The tools highly support extracting knowledge about the functioning of legacy applications and storing them in the form of precise requirement-level models. What is more, the tools enable automated transformation of these models into object-oriented code, compliant with new system structure. In the demonstration we present a recovery engine, a requirements-level editor and a transformation engine in the context of a case study example.

1 Introduction and concept

For many organizations, the transition of their legacy business applications to the new architectural patterns becomes problematic. Software systems introduced many years ago are often characterized by complex monolithic structure, technologies with non-common gateways, poor interoperability and lack of support. This makes refactoring to the new structure (eg. component- or service-based architecture) or integration with other enterprise applications virtually impossible.

This demonstration presents an approach where legacy knowledge can be extracted from any existing system and migrated to a new system by determining the observable behaviour and stored in the form of requirements-level models. Figure 1 shows an overview of the process and tools. The recovery phase starts by "ripping" the legacy system's UI by using a GUI-ripping tool [1]. Based on this (collected XML scripts) we generate the initial requirements model (cf. RSL - see the next section) in a special TALE engine. This can then be modified (ReDSeeDS Editor) by hand to refine it or to cater for new or changed functionality. Finally we use a model transformation engine (MOLA) to generate the target system structure models (both platform independent and platform specific) and code.

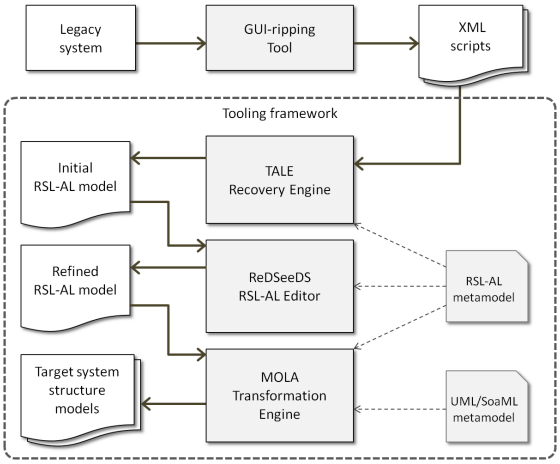

Fig. 1. Overview of the migration process

2 Tool details

The individual steps of the recovery and migration process are illustrated in Figures 2-4. The first two steps involve running the legacy system and recording the user-system interaction threads. This is done using a standard commercial test development tool and a specially developed Tool for Application Logic Extraction (TALE). The test tool records individual interactions (see labels 1-4 in Fig. 2) and form contents. This data is then translated with TALE into typical use case scenarios (see textual steps 1-4 in Fig. 2) written in formally defined Requirements Specification Language [2] (RSL).

The scenarios are accompanied by the definitions of notions used in individual sentences. These notions are also recovered by analysing the legacy UI (the forms with their fields). The recovered requirements model (use case scenarios and domain notions) can be updated to accommodate for possible changes to the system's application logic and domain. This can be done using the ReDSeeDS Engine [3] and its sophisticated RSL editor, as illustrated in Figure 3.

The final step is to generate the architectural models in UML with associated method bodies for the application logic and presentation layers. This is illustrated in Figure 4 which shows one of the transformation rules for a transformation into the Model-View-Controller/Presenter code structure. One of the rules specifies to transform every use case into a controller class with specific operations referring to scenario sentences. Another set of rules defines ways to translate from subject-verb-object and conditional sentences into sequences of method calls and "if" statements in Java. It can be also noted that the centrally defined notions (e.g. "book" of "error message") play crucial role in keeping the final code coherent. All the calls in code that are generated based on the same notion refer to the same class and associated objects in Java (see e.g. "book" and

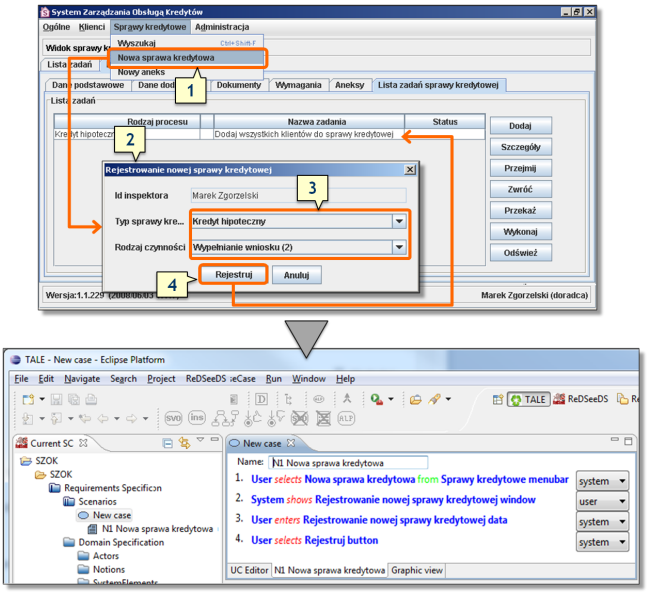

Fig. 2. Requirements-based recovery

"mBook"). Depending on the type of notion (domain or UI), the method calls refer to a model layer ("M") class objects or a view layer ("V") class objects.

The above tool suite will be demonstrated in the context of a real-life case study. It will be shown how the migration of a legacy banking system used in several Polish banks (see Fig. 2) into a new web-based architecture can be significantly facilitated.

Acknowledgement. The presented tooling framework is being developed within the REMICS project [4] (www.remics.eu).

References

1. Memon, A.M., Banerjee, I., Nagarajan, A.: GUI ripping: Reverse engineering of graphical user interfaces for testing. In: Proceedings of the 10th Working Conference on Reverse Engineering. (2003) 260–269
2. Kaindl, H., Śmiałek, M., , Wagner, P., Svetinovic, D., Ambroziewicz, A., Bojarski, J., Nowakowski, W., Straszak, T., Schwarz, H., Bildhauer, D., Brogan, J.P., Mukasa, K.S., Wolter, K., Krebs, T.: Requirements specification language definition. Project Deliverable D2.4.2, ReDSeeDS Project (2009) www.redseeds.eu.
3. Śmiałek, M., Kalnins, A., Ambroziewicz, A., Straszak, T., Wolter, K.: Comprehensive system for systematic case-driven software reuse. Lecture Notes in Computer Science **5901** (2010) 697–708 SOFSEM'10: Theory and Practice of Computer Science.
4. Mohagheghi, P., Barbier, F., Berre, A., Morin, B., Sadovykh, A., Saether, T., Henry, A., Abhervé, A., Ritter, T., Hein, C., Śmiałek, M.: Migrating Legacy Applications

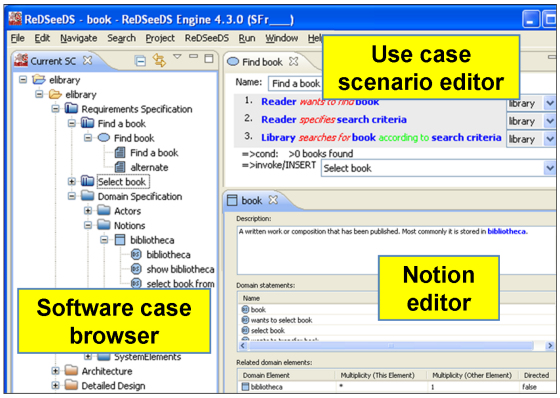

Fig. 3. Requirements level editor

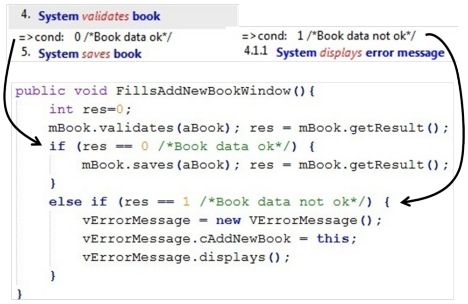

Fig. 4. Requirements-based migration to code

to the Service Cloud Paradigm: The REMICS Project. In: European Research Activities in Cloud Computing. Cambridge Scholars Publishing (2012)

The Requirements Editor RED

Harald Störrle, Maciej Kucharek

Department for Applied Mathematics and Computer Science
Technical University of Denmark
hsto@dtu.dk, kucharek.maciej@gmail.com

1 Motivation

The Requirements Editor (RED) has been conceived as a tool to support teaching a major Requirements Engineering class at the Technical University of Denmark (DTU). The course covers a wide variety of techniques in a hands-on fashion, from stakeholder analysis and goal modeling via interaction design and classic textual requirements to UML models. The need of tool support is quite obvious, but all the tools on the market covered only a small segment of these techniques, used a different (and often inconsistent) terminology, and were hard to customize. After several failed attempts to use pre-existing tools, we decided to build our own. Since this was for a RE course, we did a thorough requirements analysis up front, using the techniques taught in the course.

2 Goals, Constraints, Requirements

The primary goal of this project clearly is to support the course, in particular, to help the students understand the course material. This would, indirectly, help the teacher deliver a better course, and thus help both main stakeholders equally. As a consequence, we demand that **R1** all major topics of the course are covered, **R2**that terminology is consistent with the course material, and **R3** that students should be be freed from mechanical tasks, leaving more time for the actual course contents. Furthermore, since this is a tool that is supposed to be used by up to 70 students each year, it was necessary that **R4** the tool would run on all major platforms, with high degrees of **R5** robustness and **R6** usability. Finally, it was clear on the outset that not all desirable features would be implementable in a single increment, but instead, that many different students would be working on it over a prolonged period of time, so that **R7** maintainability was an important quality.

3 Project History

Faced with these requirements, we decided to create RED using the Eclipse Rich Client Platform (ERCP). At the time, Eclipse 3 (Indigo) was the most recent version, so that was used. A team of two students set out in late 2011 to create the tool as their joint MSc-thesis project. We created a meta-model of the

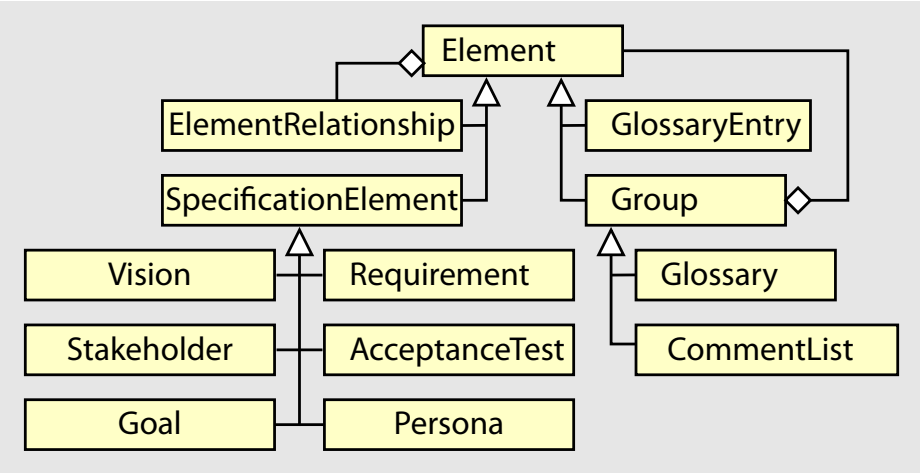

Fig. 1. Excerpt of the meta-model of RED.

concepts in RED (see Fig. 1 for an incomplete overview), and deployed a first version in September 2012.

Reactions to the first version of RED were mixed. While most students acknowledged the need for tool-support, and many saw great potential in RED, substantial shortcomings were also identified. First, RED had been tested mainly on Windows machines, and some major bugs and instabilities on MACs and some Linuxes showed only after a while. Second, RED had no facilities to support group work: the whole project is stored in one big file which conflicted with the distributed and asynchronous working style of any teams in our course. Third, the project was not structured in a way that students could easily contribute to the tool development, fixing bugs or shortcomings on-the-fly. So we have launched a re-engineering effort to address issues one and three, and new thesis projects are launched to address issue two, and completing the feature list to improve the usefulness of RED in the given context.

4 Architecture and Implementation

RED has been created using the latest Eclipse platform available at the time (Eclipse 3.7 "Indigo"). The main rationale behind for using Eclipse is its proven ability to create rich, cross-platform applications. Due to its plugin-architecture, significant leverage through reuse was expected. We also expected beneficial effects towards maintainability and long-term development by adopting a popular framework.

RED has been organized in a set of modules, each providing specific bundles of features: the Core module provides foundational contributions such as the main layout of the application and the meta-model. It supports a number of feature-modules, such as Glossary, SpecificationElements and Help. We have also

reused a number of third-party plug-ins, including EPF RichText and AgileGrid, that increased the code reuse ratio.

5 Features: done, doing, to do

RED offers currently the editors and features shown in the Fig. 2 below. Most requirements have been addressed completely or largely; some shortcomings have been identified with regards to internal software quality (which reduces maintainability), one major missing features (group-work support), and a number of minor issues regarding functionality and usability.

Current work focuses on providing functions for comparing, differencing, and merging RED-files to support group work. We expect this project to complete in the summer of 2013. Besides this, there is ongoing work to improve the on-line help function, provide a manual, and turn the project into a proper open source project to attract more contributions.

Future work will focus on visual editors for (1) goal models, and (2) models of the structure, boundaries, and collaborations of organizations and systems ("context editor"); template-driven editors to allow, e.g., to express requirements compliant with the Common Criteria and some variants of "agile" requirements (i.e., user stories), and an editor for traditional table-structured use cases, including support for project effort estimation with use case points.

Simple Editors	Complex Editors	Features
- Vision	- Personas & Scenarios	- Administrative & tracing information
- Stakeholders	* Cartoon-style	for all elements
- Goals	* Prose-style	- Flexible locking (write/comment/read)
- Glossary	* Structured text	- HTML-Report Generation
- Review Remarks	- Requirements	- Weaving of Model Fragments
- Model Structure	* Prose	- On-line help and serch functions
- Weaving Rel.s	* Test cases	- Full RTF-editor for all input texts
- Scenarios	* Model Fragments	- Enactment of scenarios

Fig. 2. Overview over the features offered in the current version of RED.

MetaEdit+: Creating Tool Support for Domain-Specific Modeling Languages

Juha-Pekka Tolvanen

MetaCase, Ylistönmäentie 31, FI-40500 Jyväskylä, Finland
jpt@metacase.com

Abstract. With MetaEdit+ you can build Domain-Specific Modeling tools and generators — without having to write a single line of code. This demonstration shows how different domain-specific languages (DSLs) can be integrated with a common metamodel and how languages can be created iteratively while automatically updating existing models.

Keywords: Domain-specific modeling, domain-specific language, language design, code generation, language workbench

1 Introduction

Domain-Specific Modeling (DSM) raises the level of abstraction and hides today's programming languages, in the same way that today's programming languages hide assembler [1]. Symbols and language constructs in a domain-specific model map to things in the problem domain. Rather than having concepts and symbols that map one-to-one with the constructs of a programming language, each symbol can be worth of several lines of code. This offers a whole level of abstraction higher than with modeling languages based on programming concepts. The developer can therefore solve the problem only once by visually modeling the solution using only familiar domain concepts. The final products can in many cases be automatically generated from these high-level specifications with domain-specific code generators [1, 2, 3].

As the name suggests, Domain-Specific Modeling is only possible because of narrowing down the design space, often to a single range of products or systems for a single company [3]. One expert defines a domain-specific language containing the domain concepts and rules, and specifies the mapping from that to code in a domain-specific code generator. Other developers then make models with the modeling language and code is automatically generated. As an expert has specified the code generators, they produce products faster and with better quality than could be done by normal developers by hand. The generated result will be free of most kinds of careless mistakes, syntax and logic errors.

Generally speaking, defining a language and generator is considered a difficult task: this is certainly true once building a language for everyone. The task eases considerably if you make it only for one problem domain in one company. This task becomes even easier if you can use tools that that support both DSM definition and use.

2 MetaEdit+ for Domain-Specific Modeling

MetaEdit+ [4] is a mature language workbench for graphical, matrix and table-based modeling languages. Originally created at the University of Jyväskylä, its core team formed a company, MetaCase, in 1991 to make it available commercially. Since 1995 MetaEdit+ has been used extensively in both industry and academia. An independent study by Eclipse modeling researchers found MetaEdit+ to be 10–50 times faster for building languages than the rest of the five tools compared [5].

With MetaEdit+ an experienced developer defines a domain-specific language in a metamodel containing the domain concepts and rules, and specifies the mapping from that to code in a domain-specific code generator. For the modeling language implementation, MetaEdit+ provides a metamodeling tool suite for defining the language concepts, rules, symbols, checking reports and generators.

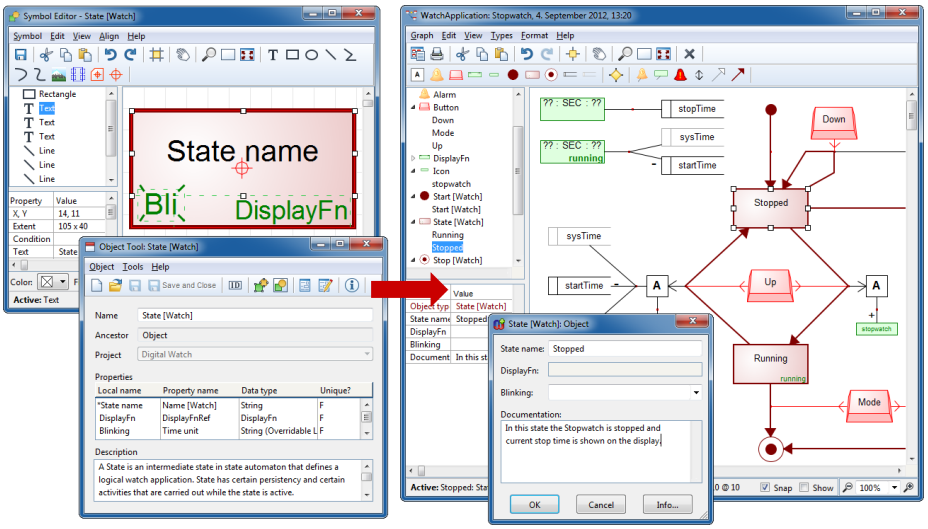

Fig. 1. Simultaneously defining the language (left) and using the language (right)

Once the metamodel is defined, or even a partial prototype, the rest of the team can start to use it in MetaEdit+. The developers make models with the modeling language and the required code is automatically generated from those models. Based on the metamodel, MetaEdit+ automatically provides modeling tool functionality such as diagramming editors, browsers, documentation generators, and multi-platform support. MetaEdit+ supports simultaneous multi-user editing via novel locking algorithms [4], as we will demonstrate.

We will show how MetaEdit+ overcomes the problems of other language workbenches by being fast — both for defining and using languages, scalable, and hiding tool implementation details. This demo will show advanced features of creating domain-specific modeling languages. We show how multiple languages can be integrated so the changes in models based on one language are visible in other models offer-

ing different views on the system specified. We also show how the tool supports language evolution, automatically updating existing models to the new version of the language. We will demonstrate how MetaEdit+ offers full debugging tools for both modeling and generators: generated code is linked back to models, so debugging can take place directly in the models.

3 Main topics and demonstration video

This demonstration addressed all the topics of EC conferences: software models (with metamodels and modeling languages), languages (with generators producing program code) and architectures (focusing on specific domains and technical platforms).

A video is available at: http://www.metacase.com/webcasts/DSM_Definition.html

4 Summary paragraph

MetaEdit+ is a mature language workbench that supports graphical diagram, matrix and table representations. Based on the language definition, MetaEdit+ automatically provides modeling tool functionality such as diagramming editors, browsers, documentation generators, and multi-platform support. We will show how MetaEdit+ overcomes the problems of other language workbenches by being fast — both for defining and using languages, scalable, and hiding tool implementation details. The demo focuses on advanced features of creating domain-specific modeling languages: how multiple languages can be integrated so the changes in models based on one language are visible in other models offering different views on the system specified and how the tool supports language evolution, automatically updating existing models to the new version of the language. We will also demonstrate how MetaEdit+ offers full debugging tools for both modeling and generators: generated code is linked back to models, so debugging can take place directly in the models.

5 References

1. Sprinkle, J., Mernik, M., Tolvanen, J-P., Spinellis, D., What Kinds of Nails Need a Domain-Specific Hammer?, IEEE Software, July/Aug (2009)
2. Kieburtz, R. et al., A Software Engineering Experiment in Software Component Generation, 18th International Conference on Software Engineering, Berlin, IEEE Computer Society Press, March, (1996)
3. Kelly, S., Tolvanen, J-P., Domain-Specific Modeling: Enabling full code generation, Wiley-IEEE Society Press (2008)
4. MetaCase, MetaEdit+ Workbench 5.0, http://www.metacase.com/support/50/manuals/ (2012)
5. El Kouhen, A., Dumoulin, C., Gerard, S., Boulet, P., Evaluation of Modeling Tools Adaptation, http://hal.archives-ouvertes.fr/hal-00706701/ (2012)

An MDE Tool-Chain for Pattern-Based S&D Embedded System Engineering -Demonstration-

A. Ziani, J. Geisel, B. Hamid

IRIT, University of Toulouse,
118 Route de Narbonne, 31062 Toulouse Cedex 9, France
{hamid,ziani,geisel}@irit.fr

Abstract. In our work, we promote a new discipline for system engineering using a pattern as its first class citizen: Pattern-Based System Engineering (PBSE). This video tutorialpresents the SEMCO MDE Tool Suite called TERESA to support PBSE in the domain of assistance to the secure and dependable embedded system engineering. We provide guidelines on how to use it to build and to store reusable artefacts (S&D patterns and property models). Once the repository is available, it serves an underlying trust engineering process [3].

1 Introduction and Motivation

The software of embedded systems is not conventional software that can be built using usual paradigms. In particular, the development of Resource Constrained Embedded Systems (RCES) addresses constraints regarding memory, computational processing power and/or limited energy. Non-functional requirements such as Security and Dependability (S&D) become more important as well as more difficult to achieve. The integration of S&D features requires the availability of both application domain specific knowledge and S&D expertise at the same time.

To tackle these challenges, we promote a new discipline for system engineering using a pattern as its first class citizen: Pattern-Based System Engineering (PBSE). In fact, capturing and providing S&D expertise by the way of patterns can support and improve embedded systems development. In this paper, we present the SEMCO MDE Tool Suite development status conducted in the context of the FP7 TERESA project aiming to support the automation of building, storing and processing reusable artefacts (S&D patterns and property models).

Using the proposed metamodels in the context of the TERESA project (http://www.teresa-project.org/) and Eclipse Modeling Framework (EMF), ongoing experimental work is done on SEMCOMDT (SEMCO Model Development Tools (http://www.semcomdt.org/), IRIT's editor and platform plugins), testing the features(see Fig. 1): (i) *Tool set A* for populating the repository, (ii) *Tool set B* for retrieval from the repository and (iii) *Tool set C* for manageming the repository.

The following details this software system from the installation, over modeling artefacts development and reuse, evolution and maintenance for acquiring organizations, end-users and front-end support provider.

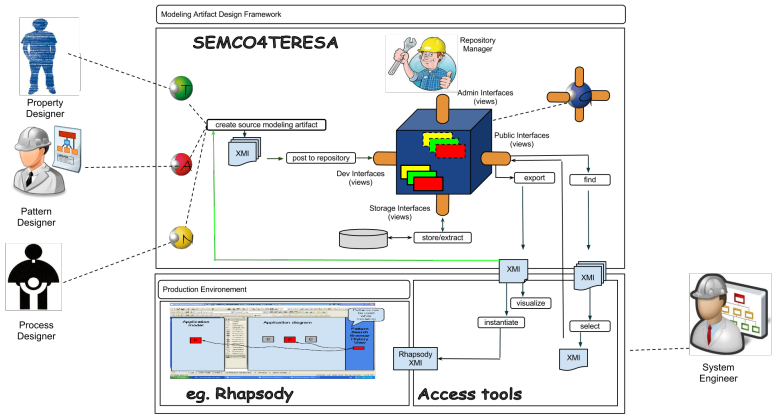

Fig. 1. Tool Architecture

2 Set-up and Links

The set up requires the installation of the SEMCO Environment which is declined into two parts. The first part is the installation and initialization of the *Gaya* CDO-based (Connected Data Objects: http://www.eclipse.org/cdo/) repository server. The second part of the SEMCO Environment is the client Tool suite, a set of modeling artefact editors and helpers to deposit to and retrieve artefacts from the repository. These tools are provided as Eclipse plugins.

- A video tutorial presenting the SEMCO MDE Tool Suite called TERESA is provided under: `http://www.semcomdt.org/semco/demo/video_semco/toolsuite/ToolSuiteIRIT.mp4`
- Tools are provided for TERESA partners as eclipse plugins: http://www.teresa-project.org/
  - As a prerequisite, the Tool suite requires Eclipse 3.7 Indigo in the Modeling Development Tools edition, augmented with Acceleo 3.2.
  - Tools (nightly builds) : `http://www.semcomdt.org/semco/tools/updates`

3 Populating

A pre-requisite task, is the creation of a new Project for the artefacts using the SEMCO Project Wizard. The first step is the specification of property model libraries for the S&D domain requirements using the *Tiqueo* EMF tree-based tool as the design editor for the modeling language defined in [2]. The next step is the creation of a unit library, setting a set of unit measures, followed by a type library and finally a category library for S&D properties. A category may need to set the type and the unit to for the measures of its instance values. In this case, these libraries need to be loaded as resources in EMF. Now, the design

validity of the result may be checked triggering the Tiqueo validation tool. If the validation succeed, the library is ready to be stored in the repository using the Deposit tool.

The *Arabion* EMF tree-based editor supports the design process of a pattern conforming to the modeling language defined in [3]. The first step contains some initialization actions to define the pattern's attributes (e.g, name, author, ...). A set of keywords may also be provided to ease the search of the pattern. To model the internal structure, representing the solution of the pattern, UML structured models are used. The pattern exposes its functions through its interfaces. The next activity is the specification of the pattern's properties. A property is typed with a category property model. The pattern designer creates a reference to the libraries previously created. Once the pattern development is completed, its design validity is checked running the Arabion validation tool. Now, the pattern is ready for repository publication, running the Arabion Deposit tool.

4 Accessing

To access the repository, three kinds of tools targeting multiple development environments on different platforms are developed: (1) Access tools are implemented as Eclipse plugin, (2) as a standalone and (3) as a web-based application. The end user or system designer uses the TERESA standalone access tool to search and to import the appropriate patterns in his development environment. The tool provides additional facilities such as pattern preview with text and diagrams and repository browsing by system development lifecycle stages. From the result list, instantiation of the appropriate ones in the development environment is done. For instance, the export allows to transform the pattern from its repository format to the Rhapsody UML format.

5 Managing

For the repository management, a set of facilities for the repository organization are provided, allowing the enhancement of its usage. Features include user, domain and artefact management. Moreover, the management of the relationships among artefact specifications and between artefact specifications and their complementary models are supported. For instance, a pattern is linked with other patterns and associated with S&D and resource property models using a predefined set of reference kinds.

References

1. Hamid, B. et al.: Model-Driven Engineering for Trusted Embedded Systems based on Security and Dependability Patterns. In: SDL Forum, LNCS pp.73–91. 2013 (To appear).
2. Ziani, A. et al.: Towards a Unified Meta-model for Resources- Constrained Embedded Systems. In: SEAA EUROMICRO Conference, IEEE pp.485–492. 2011.
3. Hamid, B. et al.: Enforcing S&D Pattern Design in RCES with Modeling and Formal Approaches. In: MODELS Conference, LNCS 6981 pp.319–333. 2011